Question 1:

1. (10 points) Determine analytically (without using computer programs) whether the following ray and triangle ABC intersect. If yes, what is the intersection point? Show your steps clearly.

- The ray emits from (1, 1, 1) and passes through the point (3, 2, -2).
- The vertices of the triangle are A = (4, 0, 0), B = (2, 2, -3), and C = (0, 4, 1).

\[
O = (1,1,1) \\
D = \text{unit vector of } (3,2,-2) - (1,1,1) \\
D = (2,1,-3)/\sqrt{14} \\
N = ((2,2,-3) - (4,0,0)) \times ((0,4,1) - (4,0,0)) = (14,14,0) \\
\text{normal of triangle plane } n = \frac{N}{|N|} = \frac{1}{\sqrt{2}}(-1,-1,0) \\
t = \frac{(P' - O) \cdot N}{D \cdot N} \\
N \cdot D = (14,14,0) \cdot \frac{2,1,-3}{\sqrt{14}} = 3\sqrt{14} \\
t = \frac{(1,1,-4) \cdot (14,14,0)}{3\sqrt{14}} = \frac{2}{3\sqrt{14}} \\
P = O + tD = (1,1,1) + \frac{2}{3}\sqrt{14} \cdot \frac{(2,1,-3)}{\sqrt{14}} = \frac{1}{3}(7,5,-3) \\
h = (P - P_o) = \frac{1}{3}(7,5,-3) - (4,0,0) = \frac{1}{3}(-5,5,-3) \\
\beta c + \gamma b = h \\
b = (P_1 - P_o) = (-4,4,1) \\
c = (P_2 - P_o) = (-2,2,-3)
\[ \mathbf{h} = P - P_0 = \frac{(13, 11, 6)}{9} (4, 0, 0) = (-23, 11, 6)/9 \]

Using xz plane
\[ \beta \cdot (-2) + \gamma \cdot (-4) = -5/3 \]
\[ \beta \cdot (-3) + \gamma = -1 \]

\[ \beta = 17/42 < 1 \]
\[ \gamma = 3/14 < 1 \]

\[ \beta + \gamma = 13/21 < 1 \]

The ray hits the triangle plane at \( \frac{1}{3} (7, 5, -3) \)
bool intersect_ray_cylinder(Cylinder cylinder, Ray& ray, double &t) {

    bool hitCylinder = false;

    // Check if Ray hits Wall:
    // Compute A, B, C, D and F coefficients
    // compute d = (radiusCap - radiusBase) * Dx
    double d = (cylinder.rc - cylinder.rb) * ray.direction.x;

    // compute a = a = Dx^2 + Dy^2 - d^2
    double Dx = ray.direction.x;
    double Dy = ray.direction.y;
    double Dz = ray.direction.z;

    double a = Dx * Dx + Dy * Dy - d * d; // dot(ray.direction, ray.direction);

    // compute F = radiusBase + (radiusCap - radiusBase) * OriginZ
    double f = cylinder.rb + (cylinder.rc - cylinder.rb) * ray.origin.z;

    // compute b = Ox*Dx + Oy*Dy - Fd
    double b = (ray.origin.x * ray.direction.x + ray.origin.y * ray.direction.y - f * d);

    // compute c
    // c = Ox^2 + Oy^2 - F^2
    double c = ray.origin.x * ray.origin.x + ray.origin.y * ray.origin.y - f * f;

    double disc = b * b - 4 * a * c;
}
// if discriminant is negative there are no real roots, so return
// false as ray misses sphere
if (disc < 0) {
    cout << "Ray does not hit cylinder wall\n";
} else {
    cout << "Ray hits cylinder wall\n";

    // finding root
    double distSqrt = sqrt(disc);
    double q;

    if (b < 0)
        q = (-b - distSqrt) / 2.0;
    else
        q = (-b + distSqrt) / 2.0;
    // compute t0 and t1
    double t0 = q / a;
    double t1 = c / q;

    // cout << "t0  " << t << endl;
    // cout << "t1  " << t1 << endl;
    // To determine if the ray hits the cylinder wall, check if centerBase ≤ z ≤ centerCap at thit.

    if (t0 > 0) {
        Point3 hit_point1 = ray.getPoint(t0);
        t = t0;
        if (cylinder.rc < hit_point1.z < cylinder.rb) {
            cout << "Hit wall:  " << hit_point1.x << "," << hit_point1.y << "," << hit_point1.z << "\n";
            hitCylinder = true;
        }
    }

    if (t1 > 0) {
        Point3 hit_point2 = ray.getPoint(t1);
        if (cylinder.rc < hit_point2.z < cylinder.rb) {
            cout << "Hit wall:  " << hit_point2.x << "," << hit_point2.y << "," << hit_point2.z << "\n";
            hitCylinder = true;
        }
    }

    if (t0 < 0) {
        t = t1;
        return true;
    } else {
        t = t0;
        return true;
    }
vector from base's center to cap's center is the normal vector

Vec3 normal = cylinder.cc - cylinder.cb; //normal to base and cap

Plane _base_plane(cylinder.cb, normal); //plane base
bool hit_base_plane = intersect_ray_plane(_base_plane, ray, t);

if (hit_base_plane) {
    if (t > 0) {
        Point3 hit_point3 = ray.getPoint(t);

        if ((hit_point3.x * hit_point3.x + hit_point3.y) < pow(cylinder.rc, 2)) {
            cout << "Intersects at base" << "t" << hit_point3.x << "," << hit_point3.y << ";" << hit_point3.z << ";" << hit_point3.z;

            hitCylinder = true;
        }
    }
}

Plane cap_plane(cylinder.cb, -normal); //cap base
bool hit_cap_plane = intersect_ray_plane(cap_plane, ray, t);
if (hit_cap_plane) {
    if (t > 0) {
        Point3 hit_point4 = ray.getPoint(t);

        if ((pow(hit_point4.x, 2) + pow(hit_point4.y, 2)) < pow(cylinder.rc, 2)) {
            cout << "Intersects at cap" << "t" << hit_point4.x << "," << hit_point4.y << "," << hit_point4.z << "," << hit_point4.z;

            hitCylinder = true;
        }
    }
}
return hitCylinder;

//Test program for cylinder intersection

int main()
{
    Ray r;

    r = Ray ( Point3 ( 0, 0, 0 ), Vec3 ( 1, 1, 1 ) );
    // cout << r.direction << endl;
    double t;
    Cylinder cylinder(Point3(0, 0, 0), Point3(0, 1, 0), 2.0, 2.0);

    bool it = intersect_ray_cylinder(cylinder, r, t);

    if (it) {
        cout << "Hit cylinder!" << endl;
        Point3 hit_point = r.getPoint(t);
        cout << "hit Cylinder point = " << hit_point << endl;
    } else
        cout << "Miss Cylinder!" << endl;
    return 1;
}
Question 3

```c
void RenderWithGlut(void) {

    glBegin(GL_POINTS);
    glVertex3f(1.0, 0.0, 0.0);
    glColor3f(1.0, 0.0, 0.0);
    glEnd();

    if (glutDraw == 0) {
        glutDraw = new GlutRenderer(); // Needs to be constructed *after* glut initialized.
    }
    // Set camera position
    glutDraw->SetupCameraView(*MainView, 1.0, MAX_DIST);
    // Define all light sources
    int i;
    for (i = 0; i < NumLights; i++) {
```
glutDraw->AddLight(*(LightArray[i]));
}
glutDraw->SetGlobalAmbientLight(GlobalAmbientR3);
glutDraw->SetGlobalAmbientLight(0.5, 0.5, 0.5);

// Render each ViewableObject
for (i = 0; i < NumObjects; i++) {
    glutDraw->RenderViewable(*(ViewObj[i]));
}
/

//drawing the ray
Vec3 D = generateRandomDirection();
glBegin(GL_LINES);
glVertex3f(Orrigin.x, Orrigin.y, Orrigin.z);
glVertex3f(Orrigin.x + dist * D.x, Orrigin.y + dist * D.y, Orrigin.z + dist * D.z);
glEnd();

//check intersection
//ray vector
glColor3f(1.0, 0.0, 0.0);
Point3 o(Orrigin.x, Orrigin.y, Orrigin.z), d(Orrigin.x + dist * D.x, Orrigin.y + dist * D.y, Orrigin.z + dist * D.z);
Vec3 diff = d - o;
Cylinder cy = Cylinder(Point3(4, 6, 0), Point3(4, 6, 1), 2.0, 2.0);
double di = sqrt(diff.x * diff.x + diff.y * diff.y + diff.z * diff.z);

//Ray r = Ray (Point3 (1, 0, 0), Vec3 (0, 3/sqrt(34), -5/sqrt(34)));
Ray r = Ray(o, diff * (1.0 / di));

    cout << "direction of ray:" << r.direction << endl;
    double t;
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(5.0);
    Triangle triangle(Point3(7, 2, -6), Point3(5, 4, -6), Point3(4, 3, -7));
    bool it = intersect_ray_triangle(triangle, r, t);
    if (!it)
        cout << "Miss Triangle!" << endl;
    else {
        cout << "Hit Triangle!" << endl;
        Point3 hit_point = r.getPoint(t);
        hitPoints.push_back(hit_point);
        cout << " hit Triangle point = " << hit_point << endl;
    }
// check hitting sphere
double rs = 1.0;
Sphere s(Point3(2, 3, -5), rs);

// r = Ray (Point3 (6, 7, 8), Vec3 (-6, -6, -8));
// cout << r.direction << endl;

it = intersect_ray_sphere(s, r, t);

if (it) {
    cout << "Hit Sphere!" << endl;
    Point3 hit_point = r.getPoint(t);
    hitPoints.push_back(hit_point);
    cout << " hit Sphere point = " << hit_point << endl;
} else
    cout << "Miss Sphere!" << endl;

// check hitting cylinder

Cylinder cylinder(Point3(4, 6, 0), Point3(4, 6, 1), 2.0, 2.0);
Cylinder cylinder(Point3(0, 0, 0), Point3(0, 1, 0), 2.0, 2.0);

it = intersect_ray_cylinder(cylinder, r, t);

if (it) {
    cout << "Hit cylinder!" << endl;
    // cout << "t = " << t << endl;
    Point3 hit_point = r.getPoint(t);
    //hitPoints.push_back(hit_point);
    cout << " hit Cylinder point = " << hit_point << endl;
} else
    cout << "Miss Cylinder!" << endl;

renderHitPoints();
/
******************************************************************************/
glutDraw->FinishRendering();
/

} void renderHitPoints() {
    for (int i = 0; i < hitPoints.size(); i++) {
        glBegin(GL_POINTS);
        glVertex3f(hitPoints[i].x, hitPoints[i].y, hitPoints[i].z);
        glColor3f(1, 0.0, 0.0);
        glEnd();
    }
}
void SetupRayTraceWorld() {
    // Initialize Pixel Array and main viewpoint
    SetUpMainView(); // in RayTraceData.cpp, declaring variable 'MainView' and creating new instance also
    // MainView is declared as extern in RayTraceData.h: CameraView *MainView
    // CameraView class defined in Graphics/CameraView.h
    // pixels = new PixelArray(640,480);
    // Array of pixels
    pixels = new PixelArray(800, 800);
    // Array of pixels
    MainView->SetScreenPixelSize(*pixels);
    float color[] = {0.1, 0.8, 0.1};
    SceneDescription scene;
    scene.SetBackGroundColor(color);
    // at RaytraceMgr/SceneDescription.h
    // Initialize Array of Materials
    //SetUpMaterials();
    // Initialize Array of Lights
    SetUpLights();
    SetUpMaterials();
    // Initialize array of viewable objects
    SetUpViewableObjects();
    const int MAX_OBJECTS = 8;

    //sphere
    ViewableSphere* vs = new ViewableSphere;
    // vs->SetCenter(6.5, 0, 12);
    vs->SetCenter(2.3, -5);
    vs->SetRadius(1.0);
    //MatArray defined in RayTrace/RayTraceData.cpp
    vs->SetMaterial(MatArray[1]);
    ViewObj[0] = vs;

    //cylinder
    ViewableCylinder *cylinder = new ViewableCylinder;
    cylinder->SetHeight(2.0);
    cylinder->SetRadius(2);
    // cylinder->SetCenter(1, 1, 2);
    cylinder->SetCenter(4, 6, 0);
    cylinder->SetMaterial(MatArray[3]);
    ViewObj[2] = cylinder;
/triangle

ViewableTriangle *triangle = new ViewableTriangle;

const VectorR3 *vertexA = new VectorR3(7, 2, -6);
const VectorR3 *vertexB = new VectorR3(5, 4, -6);
const VectorR3 *vertexC = new VectorR3(4, 3, -7);
// Point3 (2, 2, 2), Point3 (4, 6, 7), Point3 (3, 4, 6)
triangle->Init(*vertexA, *vertexB, *vertexC);
triangle->SetMaterial(MatArray[1]);
ViewObj[1] = triangle;

NumObjects = 3;
assert(NumObjects <= MAX_OBJECTS);
}

// set up light
void SetUpLights() {
    // Global ambient light and the background color are set above.

    // Initialize Array of Lights
    LightArray[0] = new Light();
    LightArray[0]->SetColorAmbient( Lt0amb );
    LightArray[0]->SetColorDiffuse( Lt0diff );
    LightArray[0]->SetColorSpecular( Lt0spec );
    LightArray[0]->SetPosition( Lt0pos );

    NumLights = 1;
    assert( NumLights <= MAX_LIGHTS );
}

// Lighting values
float Lt0amb[3] = {0.0f, 0.0f, 0.0f};
float Lt0diff[3] = {1.0f, 1.0f, 1.0f};
float Lt0spec[3] = {1.0f, 1.0f, 1.0f};
float Lt0pos[3] = {9.0, 9.0f, 0.0f};

Question 4:
/Users/Tuan_Nguyen/Library/Developer/Xcode/DerivedData/62...
float x;
float y;

vector<Point> pointSet;
int counter = 0;

struct less_than_key
{
    inline bool operator()(const Point& point1, const Point& point2)
    {
        return (point1.x < point2.x);
    }
};

//initialization
void init(void)
{
    glClearColor(0.0, 0.0, 0.0, 0.0);  //get white background color
    glColor3f(0.6f, 1.0f, 0.6f);  //set drawing color
    glPointSize(4.0);  //a dot is 4x4
    glMatrixMode(GL_PROJECTION);  //subsequent calls affect projection matrices
    glLoadIdentity();  //replace current matrix with identity matrix
    gluOrtho2D(0.0, 500.0, 0, 500);
}

void GeneratePointSet(int n, Point P1, Point P2, int high) {
    if(n == 0) {
        return;
    }
    else {
        float xmid = (P2.x + P1.x)/2;
        int temp = rand()%(high);
        float yrand = P1.y + (float)temp;
        Point Pmid(xmid, yrand);
        Point newPoint(xmid, yrand);
        pointSet.push_back(newPoint);
        counter++;  
        GeneratePointSet(n-1, P1, Pmid, high/2);
        GeneratePointSet(n-1, Pmid, P2, high/2);
    }
}

void display(void)
{
    glClear( GL_COLOR_BUFFER_BIT );
    Point P1(0,0);
    Point P2(500,0);
    GeneratePointSet(7,P1,P2,300);
    //sort
    std::sort(pointSet.begin(), pointSet.end(), less_than_key());

    //rearrange the second portion
    int count = counter/2;
    int backCount = counter;
    int temp;
    for(int i = counter/2; i < counter - counter/4; i++) {
        temp = pointSet[count].y;
        pointSet[count].y = pointSet[backCount].y;
    }
}
pointSet[backCount].y = temp;
count = count + 1;
backCount = backCount - 1;
}

//render
glBegin( GL_POLYGON );
glVertex2d(0,-500);
// glVertex2d(0,0);
for (int i = 0; i < counter; i++) {
    glVertex2d(pointSet[i].x, pointSet[i].y);
}
glVertex2d(500,0);   glEnd();
glFlush();    //send all output to screen
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);    //initialize toolkit
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB );
    glutInitWindowSize(500, 500);    //set window size on screen
    glutInitWindowSize(100, 150 );    //open screen window
    glutCreateWindow(argv[0]);
    init();
    glutDisplayFunc (display);    //points to display function
    glutMainLoop();    //go into perpetual loop
    return 0;
}
Question 5:
Let $R_1$ and $R_2$ be the two rotations with matrix representation given below:

$$R_1 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \quad R_2 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Quaternion to rotation matrix representation:

$$R_1 = \begin{pmatrix} \frac{a^2 + b^2 - c^2 - d^2}{2} & \frac{2bc - 2ad}{2} & \frac{2ac + 2bd}{2} \\ \frac{2ad + 2bc}{2} & \frac{a^2 - b^2 + c^2 - d^2}{2} & \frac{2cd - 2ab}{2} \\ \frac{2bd - 2ac}{2} & \frac{2ab + 2cd}{2} & \frac{a^2 - b^2 - c^2 + d^2}{2} \end{pmatrix}$$

$q = a + bi + cj + dk$

Finding Quaternion representation for $R_1$:

$$2ac + 2bd = 1$$
$$a^2 - b^2 + c^2 - d^2 = 1$$
$$2bd - 2ac = -1$$
$$2ad + 2bc = 0$$

$$a = -\frac{1}{\sqrt{2}}, b = 0, c = -\frac{1}{\sqrt{2}}, d = 0$$

or

$$a = \frac{1}{\sqrt{2}}, b = 0, c = \frac{1}{\sqrt{2}}, d = 0$$

$$q = -\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j$$

or

$$q = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}j$$

Finding Quaternion representation for $R_2$:...
\[2bc - 2ad = -1\]
\[2ad + 2bc = 1\]
\[a^2 - b^2 - c^2 + d^2 = 1\]
\[2bd - 2ac = 0\]

\[a = -\frac{1}{\sqrt{2}}, b = 0, c = 0, d = -\frac{1}{\sqrt{2}},\]

and

\[a = \frac{1}{\sqrt{2}}, b = 0, c = 0, d = \frac{1}{\sqrt{2}},\]

\[q = -\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}k\]

and

\[q = \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}k\]

\[v = (1, 3, 2).\]

\[R_{v,v}:\]

\[f(v) = qvq^{-1} = \left(\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}j\right)(0,1,3,2)\left(\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}j\right) = (-3,1,2)\]

\[R_{v,v}:\]

\[f(v) = qvq^{-1} = \left(\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}k\right)(0,1,3,2)\left(\frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}}k\right) = (2,3,-1)\]
Question 6:
/reusing the anaglyph.cpp

/*
 * Creating a wire house with a roof and 1 door
 */

void MakeHouse(void)
{
  XYZ pmin = {-3,-3,-3},pmax = {3,3,3};

  glBegin(GL_LINE_STRIP);
  glVertex3f(pmin.x,pmin.y,pmin.z);
  glVertex3f(pmax.x,pmin.y,pmin.z);
  glVertex3f(pmax.x,pmin.y,pmax.z);
  glVertex3f(pmin.x,pmin.y,pmax.z);
  glVertex3f(pmin.x,pmin.y,pmin.z);
  glVertex3f(pmin.x,pmax.y,pmin.z);
  glVertex3f(pmax.x,pmax.y,pmin.z);
  glVertex3f(pmax.x,pmax.y,pmax.z);
  glVertex3f(pmin.x,pmax.y,pmax.z);
  glVertex3f(pmin.x,pmax.y,pmin.z);
  glVertex3f(0.0,5.0,0.0);
  glVertex3f(pmax.x,pmax.y,pmax.z);
  glVertex3f(0.0,5.0,0.0);
  glVertex3f(pmin.x,pmax.y,pmax.z);

  glEnd();
glVertex3f(0.0, 5.0, 0.0);
glVertex3f(pmin.x, pmax.y, pmin.z);
glVertex3f(0.0, 5.0, 0.0);
glVertex3f(pmax.x, pmax.y, pmin.z);
glEnd();

glBegin(GL_LINE_STRIP);
glVertex3f(1.0, pmin.y, -3.0);
glVertex3f(1.0, pmin.y+3, -3.0);
glVertex3f(1.0 - 2, pmin.y+3, -3.0);
glVertex3f(1.0 - 2, pmin.y, -3.0);

glEnd();

glBegin(GL_LINES);
glVertex3f(pmax.x, pmin.y, pmin.z); glVertex3f(pmax.x, pmax.y, pmin.z);
glEnd();
glBegin(GL_LINES);
glVertex3f(pmax.x, pmin.y, pmax.z); glVertex3f(pmax.x, pmax.y, pmax.z);
glEnd();
glBegin(GL_LINES);
glVertex3f(pmin.x, pmin.y, pmax.z); glVertex3f(pmin.x, pmax.y, pmax.z);
glEnd();
}
Question 7:

[Image of a 3D model of a building with multiple floors and windows.]

Question 8:

[Image of a 3D modeling software interface with a car model and a character standing next to it.]